☜Infinity Limited☞ Constraining a Constraint-Based Theory

Daniel Currie Hall • University of Toronto • February 2000

Phonology generals paper

Advisor:	William J. Idsardi (University of Delaware)
Committee:	B. Elan Dresher
	Keren D. Rice

This paper describes an algorithm for computing optimal outputs without generating an infinite set of candidate forms. It offers examples that show how the algorithm can produce the same output forms as standard Optimality Theory, and explores some of the ways in which eliminating the infinite candidate set imposes useful restrictions on the theory.

ACKNOWLEDGMENTS: The research presented here was partially supported by SSHRC grant #410-96-0842 to Elan Dresher and Keren Rice. I would like to thank Bill Idsardi, Elan Dresher, and Keren Rice for their valuable comments, suggestions, and insights. I am also grateful to audiences at the first Mid-Continental Workshop on Phonology, the eighteenth West Coast Conference on Formal Linguistics, and the 1999 meeting of the Canadian Linguistic Association for comments and questions on earlier versions of this work. Of my fellow students, I would especially like to thank Susana Béjar and the members of the phonology group for their advice and encouragement. Finally, thanks to Elizabeth and Antonio for providing an extra incentive to get this thing written up, and apologies to Infinity, Ltd., the Princeton University science fiction society, for appropriating their name. Any and all errors are, of course, deliberately included for the amusement of the careful reader.

Optimality Theory, as formulated by Prince and Smolensky (1993), is based on the notion that for each underlying representation in a language, all possible surface forms are evaluated against a set of ranked, violable constraints, with the most harmonic form being selected. While there is evidence to suggest that violable constraints are part of the apparatus of the language faculty, the generation and comparison of an infinite number of candidate forms is clearly beyond the computational capacity of the human brain. In this paper, I propose an algorithm for the calculation of optimal forms. Although this algorithm does not involve direct competition or comparison between candidate forms, it is faithful in spirit and in operation to the patterns of constraint interaction described by Prince and Smolensky. In addition, I argue that the limitations imposed upon Optimality Theory by the proposed algorithm lead the theory to make stronger predictions.

This attempt to make optimality computable is not the first; however, it differs in significant ways from earlier proposals. Previous approaches include Ellison (1995), Hammond (1997), Karttunen (1998), Frank and Satta (1998), Eisner (1999), and Heiberg (1999). The main focus of this line of inquiry so far has been the implementation of Optimality Theory in computer models (Hammond; Heiberg) and the exploration of the mathematical properties of the theory (Karttunen; Frank and Satta), rather than, as in the present case, the use of computability to restrict the theory. Frequently (as in the cases of Hammond and Karttunen), the authors consider primarily a subset of phonological problems—such as the assignment of syllable structures—in which the candidate forms are easily limited to a finite number. In operation, the algorithm proposed here is similar to the method of 'lenient composition' proposed by Karttunen. In its goal of imposing limits on the theory, the current proposal is most similar to that of Eisner. However, while Eisner is concerned with the mathematical formulation of constraints, the present paper deals with the empirical consequences of eliminating the candidate set and restricting the possible kinds of constraint rankings.

The organization of the paper is as follows: Section 1 explains the motivation for creating a finite version of Optimality Theory. Section 2 provides a formulation of the proposed algorithm and explains its operation. Section 3 examines the algorithm's potential consequences for the theory, which include the elimination of sympathy constraints (as in McCarthy 1998, Karvonen and Sherman 1997, de Lacy 1998) and equally ranked or tied constraints (Truckenbrodt 1997, Broihier 1995, Pesetsky 1997, among others). Section 4 describes an attempt at implementing the algorithm in a computer program; the code of the program is given in an appendix. Finally, section 5 offers some concluding remarks about the value of a restrictive theory.

1. Why finiteness?

Prince and Smolensky (1993) maintain that computability should not be a goal of Optimality Theory. "It is not incumbent upon a grammar to compute," they write (in a section devoted to potential 'qualms' about Optimality Theory), "as Chomsky has emphasized repeatedly over the years. A grammar is a function that assigns structural descriptions to sentences; what matters formally is that the function is well-defined" (Prince and Smolensky 1993: 197). It is true that a descriptively adequate model of a language need only produce the same structural descriptions as an adult speaker of that language, and do so in a formally well-defined manner. However, a grammar that depends upon the generation and evaluation of an infinite number of candidate forms is well-defined only at the most abstract level; it cannot be implemented in any machine, including the human brain, without first being translated into some finite algorithm. If we are to attain explanatory adequacy, then our model of any language must be capable of arising in a human mind by the exposure of that mind's Universal Grammar to data from the language in question. It is incumbent upon both the initial state of the speaker's competence and all subsequent states to compute, and to do so using only the resources available in the speaker's mind. Since the processing of an infinite number of forms requires either an infinite number of processors or an infinite amount of time, neither Gen nor Eval as Prince and Smolensky describe them can be part of Universal Grammar or of the grammar of any language.

As Prince and Smolensky observe, the computational capability of the human mind is as yet dimly understood, and at this point in the history of cognitive science it is impossible to say precisely what constitutes a computationally realistic model of human language. In their view, "it is pointless to speak of relative degrees of failure: as a failed image of psychology it hardly matters whether a device takes twice as long to parse five words as it takes to parse four words, or a thousand times as long" (Prince and

Smolensky 1993: 198). There is, however, an obvious and meaningful difference between a device that takes arbitrarily much time to parse four words and a device that takes infinitely much time to parse four words. While it is too early to argue for computational efficiency, it is not too early to insist on computational possibility.

Requiring computability also has the useful effect of restricting the power of Optimality Theory. Of all the qualms Prince and Smolensky address, they are most concerned about the charge that a constraint-based theory may be insufficiently restrictive—that it predicts the existence of more types of language than are actually permitted by Universal Grammar. However, they insist that their system of ranked (rather than weighted) constraints is already adequately restrained, and that computability would be an arbitrary requirement, a restriction imposed for restriction's sake alone. Obviously, any restriction on the theory must have more specific motivation than a vague desire for limits. I have just argued that computability is necessary if Optimality Theory is to aspire to explanatory adequacy; in section 3, I will show that the restrictions that result from imposing computability on the theory have independent motivation. In particular, the algorithm I propose in this paper rules out sympathy constraints (as proposed by McCarthy (1998)) and imposes restrictions on constraint rankings—equally ranked constraints such as those used by Truckenbrodt (1997) and tied constraints as proposed by Pesetsky (1997) are not allowed. Prohibiting sympathy reduces the potential for chaotic effects in the grammar (see Idsardi 1997, 1999) and compels Optimality Theory to hold to the strong predictions inherent in its surface-oriented nature. Disallowing tied or equally ranked constraints limits the number of possible constraint rankings and thereby reduces the task of the language learner. These consequences are explored more fully in section 3.

2. Finiteness how?

The algorithm I propose is shown in (1). The steps shown here represent the calculation of an output form given a constraint ranking and an input; while the algorithm itself does not explicitly refer to the input, it invokes all the constraints in the grammar, including faithfulness constraints, which do refer to the input.

(1)

1. Let E be the ranked list of constraints.
2. Let R be a set of ordered pairs ⟨*c, v*⟩, where *c* represents a constraint and *v* a non-negative integer. A form *f* is said to satisfy R iff for every pair ⟨*c, v*⟩ in R, *f* contains no more than *v* violations of *c*.
3. Let the initial value of R be {⟨'The candidate is an interpretable linguistic entity,' 0⟩}.
4. Let *n* be an integer with an initial value of zero.
5. Let *e* represent the highest-ranked constraint in E.
6. Attempt to construct a form that satisfies R and contains exactly *n* violations of *e*.
7. If this is impossible: Increase the value of *n* by 1 and go to step 6.
   Otherwise:
8. Add the pair ⟨*e, n*⟩ to R and delete *e* from E.
9. If E is not empty: Go to step 4.
   Otherwise:
10. Any form that can be constructed so as to satisfy R is optimal.

The algorithm can be characterized in less formal terms as a combination of Gen and Eval. The generative mechanism begins as a fully unrestricted generator of linguistic forms. One by one, the constraints are incorporated into it, from highest-ranked to lowest-, as restrictions on its ability to generate. Since constraints in Optimality Theory are violable and sometimes conflict with one another, the incorporation of a constraint into the mechanism will sometimes contradict an existing restriction, and the generator will be unable to produce anything. When this happens, the most recently added constraint is relaxed so that forms that violate it exactly once may be produced. If no such forms are possible, the constraint is further relaxed, one more violation being permitted at a time, until the generator can once again generate. Once all constraints have been incorporated into the generator, anything and everything it can produce is by definition optimal.

The algorithm proceeds serially, and its operation can thus be described as derivational. However, while a traditional derivation starts with the input form and applies a series of transformations to it, the algorithm starts with a fully unspecified form and applies a series of restrictions to it. In a traditional derivation, the effects of one rule may be undone by a later rule—this is the 'Duke of York gambit' described by Pullum (1976) and deplored by McCarthy (1998). The algorithm in (1), on the other hand, proceeds monotonically from an unspecified form to a specified one, because its ability to backtrack is confined to whatever constraint it is attempting to incorporate at any given iteration. A newly incorporated constraint cannot undo the restrictions imposed by a higher-ranking constraint; it can only impose further restrictions of its own. At any point in the derivation, only the most recently incorporated constraint can be relaxed; therefore, as in conventional Optimality Theory, a lower-ranking constraint may be violated for the sake of satisfying a higher-ranking one, but not vice versa. By taking constraints in series, the algorithm enforces the transparency predicted by the original formulation of Optimality Theory.

As an illustration of how the algorithm treats a typical OT calculation, consider a constraint-based treatment of vowel epenthesis between prepositions and their objects in Czech.[1] An epenthetic [e] is inserted to break up sequences of consonants that are identical in continuancy and place of articulation, as shown in (2).

(2)

a. Epenthesis:

i.	*ke Karlovi*	‘to Karel’s place’	
	/k+karel+ovi/	→	[kekarlovi]
	to+Karel+m.dat.sg.		
ii.	*se zvonem*	‘with a bell’	
	/s+zvon+em/	→	[sezvonem]
	with+bell+m.inst.sg.		

b. No epenthesis:

i.	*k Tomášovi*	‘to Tomáš’s place’	
	/k+toma:ʃ+ovi/	→	[ktoma:ʃovi]
	to+Tomáš+m.dat.sg.	(different place of articulation)	
ii.	*s tvářem*	‘with a face’	
	/s+tva:ř+em/	→	[stva:řem]
	with+face+m.inst.sg.	(different continuancy)	

This pattern readily lends itself to a constraint-based analysis: epenthesis occurs only for the sake of avoiding ill-formed sequences, and its interactions with other phenomena (such as voicing assimilation) are transparent. (In rule-based terms, epenthesis bleeds voicing assimilation: thus in (2.a.ii.) we

[1]The conditions for epenthesis are simplified here for the sake of providing a straightforward illustration.

find [sezvonem] rather than [*zezvonem]. The conditions for voicing assimilation in Czech are described in Hall (1998).)

In somewhat simplified terms, the phenomenon of epenthesis can be accounted for using the following five constraints:

(3)

MAX	-	All input segments have output correspondents
IDENT	-	All output segments are identical to their input correspondents
*C_iC_i	-	No consonants of identical place and continuancy may be adjacent in the output
DEP	-	All output segments have input correspondents
X≈e	-	All output segments are [e]

MAX, DEP, and IDENT are standard faithfulness constraints as formulated in correspondence theory (McCarthy and Prince 1995). The constraint *C_iC_i defines the class of consonant clusters to be avoided; it is essentially a subcase of the Obligatory Contour Principle. Finally, the constraint X≈e is an abbreviation for a series of markedness constraints whose ranking in Czech makes [e] the least marked segment. X≈e therefore determines the quality of epenthetic segments.

The crucial rankings among these constraints are shown in (4):

(4)

*C_iC_i >> DEP	-	Epenthesis may be used to break up bad clusters
MAX >> DEP	-	Epenthesis is preferred over deletion
IDENT >> DEP	-	Epenthesis is preferred over dissimilation
IDENT >> X≈e	-	Only epenthetic segments are required to be [e]

All three faithfulness constraints are potentially in conflict with the markedness constraint *C_iC_i—an ill-formed consonant cluster could be avoided by a violation of DEP (epenthesis), a violation of MAX (deletion of one of the offending segments), or a violation of IDENT (changing the features of one of the offending segments). Ranking DEP below *C_iC_i ensures that epenthesis is a permissible strategy for breaking up ill-formed clusters; ranking DEP lowest of the faithfulness constraints ensures that epenthesis is the preferred strategy.[2] The other markedness constraint, X≈e, conflicts only with IDENT; since IDENT is ranked higher, X≈e will affect only segments with no input correspondents. An ordinary OT tableau for the calculation of *se zvonem* is shown in (5):

(5)

/s+zvon+em/	MAX	IDENT	*C_iC_i	DEP	X≈e
☞ a. [sezvonem]				*	******
b. [suzvonem]				*	*******!
c. [szvonem]			*!		******
d. [sevonem]		*!			*****
e. [eeeeeee]		*!*****			
f. [svonem]	*!				*****

Of the losing candidates, (b) is ruled out because it uses the marked [u] as an epenthetic segment; (c) contains an ill-formed consonant cluster; (d) and (e) contain segments that are unfaithful to the features of their input correspondents; and (f) fails to represent all input segments in the output.

The calculation of *se zvonem* by the algorithm in (1) would proceed as follows. At the beginning of the procedure, the only restriction on Gen is that it may produce only interpretable linguistic (in this case, phonological) entities. We can represent the set of things Gen might produce as X_0—that is, anything consisting of zero or more segments. The first step is to incorporate the highest-ranked constraint into Gen. In this case, there are no crucial rankings among the top three constraints, so the algorithm can take them in any order. For the sake of this example, the constraints will be taken in the order in which they appear in (5), so the first constraint to be incorporated is MAX.

Once MAX is incorporated into Gen, Gen can produce only forms that contain correspondents for each of the seven segments in the input /s+zvon+em/. The set of possible forms can now be written as X_7 (at least seven segments), or, more precisely, $X_0X_sX_0X_zX_0X_vX_0X_oX_0X_nX_0X_eX_0X_mX_0$ (seven segments of undetermined quality coindexed with the segments in the input, and the possibility of additional segments at any point in the form).

[2]Again for the sake of simplicity, I ignore the possibility of using metathesis as a repair strategy.

The next constraint to be added is IDENT, which has the effect of fixing the quality of each segment that has an input correspondent. At this stage, this constraint affects only the seven required output segments; however, if a lower-ranked constraint were to introduce a new output correspondent for one of the input segments, the new output segment would be compelled to be faithful to its input correspondent. The range of possible forms can be described as $X_0s_sX_0z_zX_0v_vX_0o_oX_0n_nX_0e_eX_0m_mX_0$; however, since the input-output correspondence relationships have now served their purpose in determining the qualities of the seven non-epenthetic segments, I will leave them out of the representation and simply write $X_0sX_0zX_0vX_0oX_0nX_0eX_0mX_0$.

The algorithm then turns to the constraint $*C_iC_i$. While the preceding two constraints have made positive demands on the output form, $*C_iC_i$ enforces a negative requirement: no two consonants of identical place and continuancy may be adjacent. In this case, there is only one way to prevent the [s] and the [z] from being adjacent: since their presence and quality have already been established, at least one other segment must intervene between them. The resulting set of possible forms can be abbreviated $X_0sX_1zX_0vX_0oX_0nX_0eX_0mX_0$. Of course, $*C_iC_i$ must continue to hold throughout the rest of the calculation, so any new segments introduced must not enter into unacceptable consonant clusters—for example, it would be impossible for the calculation to introduce an [f] immediately after the [v].

Attempting to introduce the constraint DEP creates a contradiction. DEP has the effect of setting an upper limit of zero on segments with no input correspondents. However, we already know that there must be at least one (epenthetic) segment between the [s] and the [z]. Since it is impossible to have ‘at least one and at most zero’ segments, DEP must be relaxed. If exactly one violation of DEP is permitted, then the contradiction goes away: there is ‘at least one and at most one’ segment between the [s] and the [z], and no other epenthetic material. The set of possible forms is now sXzvonem.

The last constraint, X≈e, also conflicts with existing restrictions. This constraint requires that all output segments be [e]; only two of the eight output segments can comply with this requirement. The [e] corresponding to the input /e/ already obeys X≈e, and the epenthetic segment has no reason not be an [e]. However, the other six segments cannot be made to comply with the new restriction; they can be neither removed (because of MAX) nor altered (because of IDENT). Therefore, X≈e must be relaxed six times for the contradiction to be resolved.

Once the last constraint has been incorporated, anything that can be produced is optimal. In this case, there is only one possible output, [sezvonem], because the constraints suffice to specify all aspects of the output form. This is not, however, a necessary property of the algorithm. For example, if the constraint X≈e had not been part of the grammar, then the calculation would have been complete once DEP had been incorporated, and the output would be the underspecified [sXvonem], which could be realized phonetically as [sezvonem], [suzvonem], [srzvonem], or any of several other possibilities. Producing an underspecified output is one of two ways in which this algorithm can produce the effect of free variation, which is represented in conventional OT by tied candidates. (Free variation can also result from indifferently ranked constraints; this is discussed in section 3.1.)

Figure 1 summarizes the calculation of *se zvonem* described in the preceding paragraphs. The leftmost column shows the constraints incorporated so far; the next column shows the current set of restrictions on Gen; the third column shows the underspecified form representing the set of things Gen can produce; and the final column shows the next step to be taken by the algorithm. Graphically, this approach hardly represents an improvement over conventional OT, as Figure 1 is considerably more unwieldy than the compact tableau in (5). On the other hand, the tableau owes its compactness to the fact that it omits all but six of the infinite number of candidates under consideration; Figure 1 is complete in itself. In Figure 1, the elimination of candidates is implicit rather than explicit: the changing of X_0 into $X_0X_sX_0X_zX_0X_vX_0X_oX_0X_nX_0X_eX_0X_mX_0$, for example, corresponds to the elimination of [*svonem], [*zvonem], [*szvo], [*ta], and all other candidates that do not contain correspondents for all segments in the input. As the output gradually becomes more fully specified, the suboptimal candidates are eliminated by implication.

The algorithm deals equally well with the sorts of calculations to which OT is best suited—for example, the assignment of syllable structures. In the task of parsing segments into syllables, there is perhaps less need for an algorithm that does away with comparison between candidates, for although the candidate set is in principle infinite, in practice it can be made finite by our knowledge of the inventory of constraints. (For example, the number of

Figure 1. Input: /s+zvon+em/ Constraints: MAX >> IDENT >> $*C_iC_i$ >> DEP >>X≈e

	Restrictions on Gen	Current Form	Next Action
	1. The output is a phonological entity.	X_0	add first constraint
 MAX	1. The output is a phonological entity. 2. All input segments are present in the output.	$X_0X_sX_0X_zX_0X_vX_0X_oX_0X_nX_0X_eX_0X_mX_0$	add next constraint
 MAX IDENT	1. The output is a phonological entity. 2. All input segments are present in the output. 3. All output segments are identical to their input correspondents.	X_0sX_0zX_0vX_0oX_0nX_0eX_0mX_0	add next constraint
 MAX IDENT $*C_iC_i$	1. The output is a phonological entity. 2. All input segments are present in the output. 3. All output segments are identical to their input correspondents. 4. No Cs identical in place and continuancy are adjacent in the output.	X_0sX_1zX_0vX_0oX_0nX_0eX_0mX_0	add next constraint
 MAX IDENT $\underline{*C_iC_i}$ **DEP**	1. The output is a phonological entity. 2. All input segments are present in the output. 3. All output segments are identical to their input correspondents. 4. No Cs identical in place and continuancy are adjacent in the output. **5. All output segments have input correspondents.**	s$X_{\underline{1}}^{\mathbf{0}}$ zvonem	**contradiction:** relax DEP
 MAX IDENT $*C_iC_i$ DEP(*)	1. The output is a phonological entity. 2. All input segments are present in the output. 3. All output segments are identical to their input correspondents. 4. No Cs identical in place and continuancy are adjacent in the output. 5. All output segments but one have input correspondents.	sXzvonem	add next constraint
 MAX IDENT $*C_iC_i$ DEP(1*) **X≈e**	1. The output is a phonological entity. 2. All input segments are present in the output. 3. All output segments are identical to their input correspondents. 4. No Cs identical in place and continuancy are adjacent in the output. 5. All output segments but one have input correspondents. **6. All output segments are [e].**	$(\underline{s}=\mathbf{e})e(\underline{z}=\mathbf{e})(\underline{v}=\mathbf{e})(\underline{o}=\mathbf{e})(\underline{n}=\mathbf{e})e(\underline{m}=\mathbf{e})$	**contradiction:** relax X≈e (six times)
 MAX IDENT $*C_iC_i$ DEP(1*) X≈e(6*)	1. The output is a phonological entity. 2. All input segments are present in the output. 3. All output segments are identical to their input correspondents. 4. No Cs identical in place and continuancy are adjacent in the output. 5. All output segments but one have input correspondents. 6. All output segments but six are [e].	[sezvonem]	FINISHED

Key: Restrictions shown in **boldface** cannot be incorporated because they introduce contradictions.
The higher-ranking constraints that overrule them are underlined.
In the output form, *italic type* indicates representations subject to further specification.
Parts of the output that are shown in roman type may not be changed.

potentially useful epentheses is limited by the number of input segments; see Prince and Smolensky (1993: 94–96).) However, for the sake of establishing the generality of the algorithm, I will briefly show how it works in calculating syllable structures. As we shall see, the rigour of the algorithm brings out the need for greater explicitness in constraint-based grammars. In addition, applying the algorithm to problems of syllable structure demonstrates that key insights of the constraint-based approach can be stated in terms of a serial calculation.

Consider the following set of constraints, which Prince and Smolensky (1993: 160) use to illustrate their account of the fact that "there are languages in which some possible onsets are not possible codas, but no languages in which some possible codas are not possible onsets":

(6)

Parse	- Input segments must be parsed into syllables in the output
*M/□	- Epenthetic segments may not be margins (onsets or codas)
*P/d	- The segment [d] may not be a peak (nucleus)
*P/□	- Epenthetic segments may not be peaks
–Cod	- Syllables should not have codas
Ons	- Syllables should have onsets
*M/i	- The segment [i] may not be a margin
*P/i	- The segment [i] may not be a peak
*M/d	- The segment [d] may not be a margin

Note that no distinction is made between onsets and codas in the constraints *M/□, *M/i, and *M/d. Prince and Smolensky's point is that asymmetries in licensing between onsets and codas can be made to follow from the fact that the universal set of constraints includes Ons and –Cod but no constraints requiring codas or prohibiting onsets. They demonstrate that a language with the constraint ranking in (7) will tolerate [i] as an onset but not as a coda, while [d] is allowed in both margin positions.[3]

[3]The following rankings presumably follow from a universal sonority hierarchy:

*P/d >> *P/i	- Less sonorous segments are worse peaks
*M/i >> *M/d	- More sonorous segments are worse margins

(7) PARSE, *M/□, *P/d, *P/□ >> –COD >> ONS >> *M/i, *P/i, *M/d

The emergent licensing asymmetries can be seen in the tableaux in (8). Periods indicate syllable boundaries; a subscript vertical line indicates that a segment is parsed as a peak; vowels parsed as margins are shown with subscript arches; a box (□) indicates an epenthetic segment; and angle brackets set off unparsed segments.

(8) adapted from Prince and Smolensky (1993: 160)

a. /tad/	PARSE	*M/□	*P/d	*P/□	–COD	ONS	*M/i	*P/i	*M/d
☞ .ta̩d.					*				*
.ta̩.d□̩.				*!					*
.ta̩.d̩.			*!			*			
.ta̩.□d̩.		*!	*						
.ta̩.⟨d⟩	*!								
b. /tai/	PARSE	*M/□	*P/d	*P/□	–COD	ONS	*M/i	*P/i	*M/d
.ta̩i̯.					*!		*		
.ta̩.i̯□̩.				*!			*		
☞ .ta̩.i̩.						*		*	
.ta̩.□i̩.		*!						*	
.ta̩.⟨i⟩	*!								
c. /ia/	PARSE	*M/□	*P/d	*P/□	–COD	ONS	*M/i	*P/i	*M/d
☞ .i̯a̩.							*		
.i̩.a̩.						*!*		*	
.□i̩.□a̩.		*!*						*	
⟨i⟩.a̩.	*!					*			

In (8a), [d] makes such a poor nucleus that it must be parsed as a coda (*P/d >> –COD). The vowel [i], on the other hand, is sufficiently harmonic as a nucleus to escape being parsed as a coda (–COD >> *P/i), as shown in (8b). However, in (8c), [i] can be parsed as an onset to prevent the creation of an onsetless syllable (ONS >> *M/i).

The algorithm produces the same results, but without comparing candidate forms. Instead, the calculation can be described as a derivation in which each step contributes to the building of syllable structure or limits the ways in which subsequent steps may do so. The calculation in (9) shows how the algorithm deals with /tad/. (In this example I consider only the following operations: parsing an input segment as a peak or as a margin, marking an input segment as unparsed, setting a syllable boundary, and inserting an epenthetic segment. Of course, there are other ways of satisfying constraints on syllable structure; however, I am concerned here—as are Prince and Smolensky—only with operations that assign syllable structure to segmental representations, not with ones that alter segmental or featural content.)

(9) /tad/

Step	Form	Restrictions
1. PARSE	.tad.	Input segments must be parsed.
2. *M/□	.tad.	no epenthesis in margins
3. *P/d	.tad.	The [d] must not be parsed as a peak.
4. *P/□	.tad.	no epenthesis in peaks (∴ no epenthesis)
5. –COD	.ta̩d.	Cannot be satisfied—if the [d] is put into a separate syllable, it cannot be a peak (3), but neither can it be the onset to a syllable with an epenthetic peak (4), nor can it be left unparsed (1). One violation of –COD must be permitted (but later steps are not allowed to introduce additional violations). The [a] must be a peak, since making it a margin would result in a second violation.
6. ONSET	.ta̩d.	(satisfied)
7–8. *M/i, *P/i	.ta̩d.	(vacuously satisfied)
9. *M/d	.ta̩d.	Cannot be satisfied—the [d] cannot be parsed as a peak (3) or left unparsed (1). One violation must be allowed.

The calculation of /ia/ is shown in (10):

(10) /ia/

Step	Form	Restrictions
1. PARSE	.ia.	Input segments must be parsed.
2. *M/□	.ia.	No epenthesis in margins
3. *P/d	.ia.	(vacuously satisfied)
4. *P/□	.ia.	No epenthesis in peaks (∴ no epenthesis)
5. –CODA	.ia̩.	The [a] must be a peak. At this point, the [i] can be either an onset to the syllable headed by the [a] or the peak of a separate syllable.
6. ONSET	.i̯a̩.	The [i] must be parsed as an onset. Parsing it as a peak would create two onsetless syllables, since epenthetic onsets are not possible here (2).
7. *M/i	.i̯a̩.	Cannot be satisfied. The [i] must be parsed (1), and it must be an onset (6). One violation must be allowed.
8. *P/i	.i̯a̩.	(satisfied)
9. *M/d	.i̯a̩.	(vacuously satisfied)

In each of these cases, the algorithm arrives at the optimal parsing by serial calculation. Instead of directly comparing possible output forms, it rules out suboptimal ones implicitly as the structure becomes more restricted. So far, the algorithm has produced the outputs shown in Prince and Smolensky's tableau. However, it does not generate .ta̩.i̩. from /tai/:

(11) /tai/

Step	Form	Restrictions
1. Parse	.tai.	Input segments must be parsed.
2. *M/□	.tai.	No epenthesis in margins
3. *P/d	.tai.	(vacuously satisfied)
4. *P/□	.tai.	No epenthesis in peaks (∴ no epenthesis)
5. NoCoda	.tai̩.	The [i] must be a peak; parsing it as a margin is ruled out because no peak can be epenthesized after it to make it an onset (4). At this point, the [a] may be either a peak or an onset.
6. Onset	.t͜ai̩.	The [a] must be a margin, since parsing it as the peak of a separate syllable would leave no way of giving the [i] an onset (2). The [t] must also be a margin; if parsed as a peak, it would have no onset (2).
7. *M/i	.t͜ai̩.	(satisfied)
8. *P/i	.t͜ai̩.	Cannot be satisfied (5). One violation must be permitted.
9. *M/d	.t͜ai̩.	(vacuously satisfied)

The algorithm parses /tai/ as a single syllable with a branching onset. This does not reflect any empirical difference between the algorithm and standard OT; if .t͜ai̩. had been included as a candidate in Prince and Smolensky's tableau, it would have been judged optimal. In an infinite candidate set, .t͜ai̩. would of course be included; however, Prince and Smolensky did not have sufficient space on the page to show an infinite candidate set.

There is no difficulty in ruling out .t͜ai̩.; the two most likely constraints for the task are *M/a (prohibiting [a] as a margin) and *Complex, which penalizes branching within onsets, nuclei, or codas. The point of this example is not to show that Prince and Smolensky produce the wrong output; rather, it is to demonstrate how easy it is to overlook a relevant candidate (and thus a relevant constraint) in selecting a finite subset of candidates to print in a tableau. If, on the other hand, constraints are seen as rules that can introduce structure or impose restrictions on subsequent rules, then at each step in the calculation, the structure and restrictions generated so far implicitly define the set of relevant candidates. The winning candidate emerges automatically as the output form becomes more specific, and no suboptimal candidates need be explicitly enumerated.

Another point illuminated by this example is the fact that a serial calculation can produce the effects of interaction among a ranked set of preferences. The most insightful aspect of OT is its use of constraints to make generalizations about disparate processes that have similar effects on surface forms. For example, it is more concise to posit a single constraint ONS than to posit several different rules that produce onsets in various environments. What the calculations in Figure 1 and in (9–11) show is that this ability to generalize is independent of whether the grammar operates serially or in parallel.

The calculations made by the algorithm are derivations; they build output forms in a sequence of steps, each of which operates on the output of the last. However, these derivations are faithful to the original formulation of OT in that they are transparent. While the introduction of each new constraint changes the form being calculated, it can only do so by adding specifications and restrictions to the form, not by changing or removing specifications already imposed. Constraints can raise lower limits on sets of segments; they can lower upper limits; they can introduce correspondence relations; they can add featural content to underspecified segments; they can specify syllable structure. They cannot lower lower limits or raise upper limits; they cannot delete correspondence indices, features, or other structure. When a newly introduced constraint comes into conflict with existing restrictions, it is the new constraint that must be relaxed in deference to its higher-ranking predecessor. Change is always in the direction of increasing specificity: this restriction enforces the transparency that is the strongest prediction of a theory based on surface-oriented constraints.

3. The consequences of finiteness

3.1 *Sympathy*

The algorithm in (1) enforces transparency in another way as well: it does not allow for the use of sympathy constraints such as those proposed by McCarthy (1997, 1998). Sympathy constraints enforce correspondence between the output and a failed candidate. Since the algorithm in (1) does not involve the generation of a candidate set, it leaves no room for correspondence between candidates.

McCarthy proposes sympathy as a way for Optimality Theory to deal with cases of opacity—phenomena that in a rule-based theory arise from counter-feeding and counter-bleeding rule orderings. One of the examples on which McCarthy bases his theory of sympathy is the Tiberian Hebrew word [deʃe] 'grass.' In the traditional rule-based account, this form is derived from underlying /deʃʔ/ by the application, in counter-bleeding order, of epenthesis and final glottal-stop deletion:[4]

(12)

UR	/deʃʔ/
Epenthesis	deʃeʔ
ʔ-Deletion	deʃe
SR	[deʃe]

McCarthy claims that [deʃe] cannot be accounted for in traditional Optimality Theory. Epenthesis is driven by a constraint against unsyllabifiable consonant clusters, deletion by a constraint against syllable-final glottal stops. In the case of /deʃʔ/, both constraints are satisfied by the omission of the glottal stop, and so there is no need for epenthesis. The constraint rankings needed to account for the ordinary, transparent effects of epenthesis and glottal stop deletion in Tiberian Hebrew are, briefly, as follows:

• *CODA/ʔ (McCarthy's CODA-COND), which penalizes glottal stops in coda position, outranks MAXC, which penalizes omission of input consonants. Thus Eval would rather an input glottal stop be absent from the output altogether than appear in a coda.

• *COMPLEX, which penalizes unsyllabifiable consonant clusters, outranks DEPV, which penalizes output vowels that lack input correspondents. Thus the grammar is willing to suffer an epenthetic vowel rather than allow an inappropriate cluster.

• MAXC outranks DEPV, so that epenthesis, and not deletion, is the preferred means of eliminating unwanted clusters.

[4]As noted by Idsardi (1999), the underlying form is actually /daʃʔ/; however, the calculation of vowel quality is not relevant here.

• ANCHOR (Root, σ, final) is active in the grammar. This constraint requires that the output correspondent of a root-final input segment be final in some syllable. The effect of this constraint is to ensure, when epenthesis resolves a root-final consonant cluster, that the epenthetic vowel is inserted between the consonants and not after them.

The tableau in (13) shows how these constraints incorrectly select [*deʃ] instead of [deʃe]:

(13) adapted from McCarthy (1998: 22)

/deʃʔ/	*COMPLEX	ANCHOR	*CODA/ʔ	MAXC	DEPV
[deʃe]				*	*!
*☞ [deʃ]				*	
[deʃʔe]		*!			*
[deʃeʔ]			*!		*
[deʃʔ]	*!		*		

Since there is no constraint on which [deʃe] performs better than [*deʃ], no ranking of the constraints in (13) will select [deʃe]. McCarthy's solution is to introduce a new constraint, ❀MAXV$_{\text{MAXC}}$, which requires the output to contain a correspondent for each vowel in the candidate that would be optimal if MAXC were the highest-ranked constraint. In other words, the output must contain as many vowels as it would have contained if it had kept all the consonants from the input. This constraint eliminates [*deʃ], as shown in (14), making [deʃe] optimal.

(14) adapted from McCarthy (1998: 22)

/deʃʔ/	❀MAXV$_{\text{MAXC}}$	*CPLX	ANCHOR	*CODA/ʔ	MAXC	DEPV
☞ [deʃe]					*	*!
[*deʃ]	*!				*	
❀ [*deʃʔe]			*!		√	*
❀ [*deʃeʔ]				*!	√	*
[*deʃ]	*!	*!		*!	√	

Of the candidates that satisfy MAXC, the most harmonic ones are [*deʃʔe] and [deʃeʔ], so these are the sympathetic candidates (marked with ❀). Since the output is required to have as many vowels as the sympathetic candidates, [deʃe] is selected over [*deʃ].

There are several objections to be made to the sympathy approach. Idsardi (1997, 1999) takes the theory through the alarmingly tortuous steps required to make it generate a few additional forms in Tiberian Hebrew. He finds that sympathy introduces chaos into the grammar: “small changes [in constraint rankings] can cause massive changes in unrelated forms” (Idsardi 1997: 28). Sympathy constraints can interfere with the selection of transparent forms that would otherwise be simple for OT to generate. For example, in an earlier approach to [deʃe], McCarthy (1997) used AlignR$_{IO}$(Root, σ) instead of MaxC as the selector constraint. This constraint, which requires the right edge of the root to be aligned with the right edge of a syllable, selects [*deʃeʔ] as the sympathetic candidate. The sympathy constraint ❀Max$_{AR}$ then requires the output to have correspondents for as many of the segments in the sympathetic candidate as possible, with the result that [deʃe] is chosen as optimal. Idsardi (1997) shows that ❀Max$_{AR}$ interferes with the otherwise straightforward calculation of [malki] from /malk+i/: the constraint enforces faithfulness to a sympathetic candidate [*melexʔi], in which extra material has been inserted to place the final consonant of the root into a coda. The result is that [*melexʔi] emerges as not only the sympathetic candidate, but also the optimal one, by virtue of its faithfulness to itself. While the use of MaxC as a selector constraint by McCarthy (1998) avoids this particular problem, the potential for chaotic effects remains a disturbing fact about sympathy in general.

A further objection to sympathy is that it does not, as McCarthy claims, preserve the parallel character of Optimality Theory. Before the output can be determined, the sympathetic candidate must be selected; otherwise, the candidates cannot be evaluated with respect to the sympathy constraint. The tableau in (14) actually represents two calculations. First, a candidate is chosen that is optimal under the ranking MaxC >> *Complex, Anchor, *Coda/ʔ >> DepV. The winner of this calculation is then used as the sympathetic form in choosing the optimal output for the ranking ❀MaxV$_{MaxC}$ >> *Complex, Anchor, *Coda/ʔ >> MaxC >> DepV.

McCarthy maintains that the dependency of the output on the sympathetic candidate does not entail a serial process in which one is derived after the other. He compares the case of output and sympathetic forms to that of bases and reduplicants: “reduplication may involve copying the base as it has been altered by phonological processes, but this does not entail that the base undergo phonology prior to reduplication” (McCarthy 1998). However, this analogy is misleading. In evaluating a candidate form against a base-reduplicant faithfulness constraint, one need only inspect the form itself, for both base and reduplicant are contained within it. In evaluating a potential output form against a sympathy constraint, one must have access to the sympathetic candidate as well as to the form currently being evaluated. For this to be possible, the sympathetic candidate must already have been calculated.

One potential way around this objection would be to have the calculation evaluate not individual forms, but pairs of sympathetic and output forms. (This approach to sympathy is taken by Jun (1998).) Sympathetic constraints would evaluate the correspondence between partners rather than evaluating all potential outputs against one sympathetic candidate. Such a calculation is shown in (15). MaxC is broken into two constraints, MAXC$_{\text{I❀}}$, which compares the sympathetic partner in each candidate to the input, and MAXC$_{\text{IO}}$, which compares the output partner to the input. (The constraint MAXV, although shown here as a single constraint, comprises MAXV$_{\text{I❀}}$ and MAXV$_{\text{IO}}$.) In the list of candidates, sympathetic partners are shown in parentheses. For the sake of clarity, a double line demarcates each group of candidates with the same output form. Asterisks in the tableau are vertically aligned with the individual forms that incur them, but they count against the pair as a whole.

(15)

/deʃʔ/	MAXCI❀	MAXV❀O	*CPLX	ANC	*CODA/ʔ	MAXCIO	DEPV
☞ deʃe (deʃeʔ)					 *	* 	* *
deʃe (deʃʔ)			 *		 *!	* 	*
deʃe (deʃ)	 *!					* 	*
deʃʔe (deʃeʔ)				* 	 *!		* *
deʃʔe (deʃʔ)			 *	*! 	 *		*
deʃʔe (deʃ)	 *!			* 			*
deʃ (deʃeʔ)		* 			 *!	* 	 *
deʃ (deʃʔ)			 *		 *!	* 	
deʃ (deʃ)	 *!					* 	
deʃeʔe (deʃeʔe)				* *!			** **

Since each potential output form must be considered in combination with each potential sympathetic form, the number of candidates is squared. However, that number was infinite to begin with, so the computational consequences may be assumed to be moot, at least in conventional Optimality Theory. What is more important is that taking seriously the analogy with reduplication has serious empirical consequences as well. Correspondence constraints evaluate correspondences between a pair of forms; compliance or failure to comply with such constraints is a property of the pair rather than of either member independently. In the case of an input-output correspondence constraint, the input is the same for all candidates, and so the constraint can affect only the choice of output. A base-reduplicant correspondence constraint, however, can affect the selection of both base and reduplicant, because both forms are chosen in the same calculation. If the calculation of the sympathy candidate is truly simultaneous with the calculation of the output, then sympathy constraints such as MAXV❀O can contribute to the selection of the sympathy candidate. This would in turn affect the choice of the output in a manner not predicted by McCarthy's original formulation of sympathy theory. Such a situation would be precisely analogous to cases of back-copying in reduplication, in which phonological changes whose environments are met only in the reduplicant appear on the base as well. Correspondence theory clearly allows for such phenomena in reduplication; McCarthy and Prince (1995: 289–325) discuss several attested cases and argue that their theory's ability to account for them represents a major advantage over rule-based approaches to reduplication. The consequences of allowing 'back-copying' from an output to its sympathy partner are less desirable: the result is the further muddying of an already chaotic theory.

My final and broadest objection to sympathy is that it represents a retreat from the most interesting prediction of Optimality Theory. A theory based on surface-oriented constraints inherently suggests that these constraints will apply transparently. The constraints in Optimality Theory are violable, but no constraint should be violated except for the purpose of satisfying a higher-ranked constraint on the form of the output, or on its relation to the input. Sympathy theory introduces a form of indirect faithfulness that allows for the translation of opaque rule-based analyses into Optimality Theory. The output is faithful to the sympathetic candidate, which is in turn faithful to the input; thus the sympathetic candidate attains a status similar to that of an intermediate form in a derivation. In fact, in the calculation of [deʃe], the sympathetic candidate [*deʃeʔ] is identical to the crucial intermediate form in the derivation in (12).

The purpose of a new theory, such as Optimality Theory, is not merely to translate old analyses, but to provoke new ones. In invoking sympathy to deal with [deʃe], McCarthy is essentially updating an analysis of Tiberian Hebrew that has appeared in various forms in (for example) Malone (1993), Prince (1975), and Gesenius (1910): the final [e] in [deʃe] is a relic from a form in which epenthesis is necessary. This analysis may well be the correct one, but in order to find out whether this is so, we must consider alternative analyses.

Optimality Theory predicts transparency; [deʃe] is traditionally analyzed as opaque. The conflict between the two suggests two questions. Sympathy theory is an answer to the question "Can Optimality Theory be made to generate opacity?" The other question is "Can [deʃe] be described as transparent?" The first question is a question about theory; the second is a question about language.

Balcaen and Hall (1999) suggest that the answer to the second question is yes. Suppose the following candidate is added to the tableau in (13):

(16)

/deʃʔ/	*COMPLEX	ANCHOR	*CODA/ʔ	MAXC	DEPV
☞ [deʃe]					

This new candidate, [deʃe], is clearly optimal, since it violates none of the constraints shown in the tableau. What makes it superior to the [deʃe] in (13) is the fact that the [e] is now in a correspondence relation with the underlying glottal stop. This candidate sacrifices featural identity between correspondents in order to satisfy the other constraints. In derivational terms, there is neither epenthesis nor deletion; instead, the glottal stop is transformed into a vowel. This approach has the economical character of a truly optimal solution: while the features of the glottal stop cannot surface in this environment, its timing slot can be, and is, preserved, being realized with the features of the default vowel in order to be syllabifiable.

Not only does this reanalysis of [deʃe] make the form transparent, more importantly, it suggests new ways of looking at other data in the language. For example, when a glottal stop is deleted following an underlying vowel, as in (17), the vowel undergoes what has traditionally been described as compensatory lengthening.

(17) √mṣʔ 'find' Qal Imperfect 3 masc. sg. [yimṣa:]

UR	/yamṣoʔ/
Lowering	yamṣaʔ
ʔ-Deletion	yamṣa
Compensatory Lengthening	yamṣa:
(other rules)	yimṣa:
SR	[yimṣa:]

The supposedly epenthetic vowel in [deʃe], however, does not lengthen. These facts are consistent with the hypothesis that what deletes is not the entire glottal stop, but only its features, and that its timing slot is realized as a vowel—a continuation of the immediately preceding vowel, if there is one, otherwise, the default vowel [e].

In addition, it may be possible to extend this analysis beyond glottal stops. The data in (18) show evidence of alternation between [y] and [i] that could be attributed to the vocalization of a root consonant:

(18)

√pry	[pərí:]	'fruit'	[piryó:]	'his fruit'
√ḥly	[ḥŏlí:]	'sickness'	[ḥolyó:]	'his sickness'

Like /ʔ/, /y/ cannot surface as a consonant when it would be unsyllabifiable; unlike /ʔ/, /y/ has features which Tiberian Hebrew permits in syllable-final position. Thus an unsyllabifiable /y/ can surface as a high, coronal [i], while a vocalized /ʔ/ obtains its features from an adjacent vowel or by default.

Balcaen and Hall's analysis does not solve all the problems of vowel-consonant alternations in Tiberian Hebrew. However, it does provide a novel way of unifying phenomena previously thought to be separate, and it suggests new directions for research on this and other languages. (A similar analysis of the opaque behaviour of [ɹ] in Eastern Massachusetts English has been proposed by Bakovic (1998).) Moreover, its central insight is independent of the theory that provoked it: glottal stop vocalization is just as viable in a rule-based theory as it is in a constraint-based theory. McCarthy's sympathy analysis, by contrast, does not change our view of what is going on in Tiberian Hebrew; it simply offers a way of translating the old analysis into the new theory. Idsardi (1999: 1) says that "opacity is the fundamental property of human language." One way to test this statement is by seeing how much of human language can be analyzed using the most restricted version of Optimalty Theory, which assumes that opacity is not a property of human language at all.

3.2 *Equally ranked constraints*

Another, less fundamental, restriction imposed by the algorithm in (1) is that it does not allow constraints to be equally ranked. The algorithm incorporates one constraint into Gen at a time, and any constraint may be relaxed in order to satisfy a previously incorporated constraint. For this procedure to work, the grammar must always be able to process the constraints in a strict order. As formulated in (1), the algorithm cannot deal with constraints that are equally ranked, because an equally-ranked pair of constraints would have to be taken together. While it would be possible to revise the algorithm to allow equal ranking of constraints, I prefer to maintain the more restrictive version.

In the Optimality Theory literature, there is some confusion between equally ranked constraints and what I will call indifferently ranked constraints. Both of these are generally indicated in tableaux by dotted lines separating the constraints in question; however, their effects on the calculation of the output are quite distinct. Two constraints C1 and C2 are equally ranked if *neither* ranking C1>>C2 nor C2 >>C1 results in the optimal output; they are indifferently ranked if *both* C1>>C2 and C2 >>C1 result in optimal outputs. The possible situations are schematized in (19):

(19)

a. Equally ranked constraints:
B is optimal, but neither C1>>C2 nor C2 >>C1 selects it.

	CONST1	CONST2
candidate A	*	****!
☞ candidate B	* *	* *
candidate C	****	*!

b. Indifferently ranked constraints:
A is optimal under any ranking.

	CONST1	CONST2	CONST3
☞ candidate A			*
candidate B	*(!)		**(!)

c. Indifferently ranked constraints: A and B are in free variation.
Either C1>>C2 or C2 >>C1 produces a grammatical output.

	CONST1	CONST2
(☞) candidate A		*(!)
(☞) candidate B	*(!)	

Scenario (19a) cannot be reproduced using the algorithm in (1). When two constraints are equally ranked, candidates are judged by the total number of violations of both constraints they incur; thus in (19a), candidate B, which incurs a total of four violations of CONST1 and CONST2, is preferred over the other two candidates, which each incur five violations. Ranking CONST1 over CONST2 would render candidate A optimal; the opposite ranking would favour candidate C. Equally ranked constraints of this sort are used by Truckenbrodt (1997) to account for the construction of phonological phrases in Bengali.

In (19b), the three constraints are indifferently ranked because there is no evidence against any potential ranking. There is no conflict among the constraints: CONST1 and CONST3 favour candidate A, while CONST2 is equally satisfied by either candidate. Since no constraint favours candidate B over candidate A, all six possible rankings will select A as the output. A real example of indifferently ranked constraints of this sort may be seen in (4) and (5), where the relative ranking of MAX, IDENT, and *C_iC_i has no effect on the choice of epenthesis as the means of resolving illicit consonant clusters in Czech.

The tableau in (19c) shows how indifferently ranked constraints can generate free variation. In this case, there is positive evidence in favour of each ranking: both A and B are attested. Since either ranking yields a grammatical output, there is no reason to prefer one over the other.

The algorithm in (1) can deal with cases such as (19b) and (19c). When the algorithm comes to a group of indifferently ranked constraints, it can simply take them in random order. In a situation such as (19b), the choice made by the algorithm will have no effect on the output; for example, the calculation in Figure 1 would still generate [sezvonem] if the first three constraints had been processed in a different order. In (19c), the algorithm will sometimes produce candidate A and sometimes candidate B, depending on which of the two constraints is incorporated first. As mentioned earlier, this is one of two ways in which the algorithm can produce forms in free variation; the other way is by generating an underspecified output.

Pesetsky (1997) proposes yet another way in which constraints might depart from the usual pattern of strict hierarchical ordering. He defines the notion of *tied constraints* as follows:

(20)
Constraint Tie: The output of a set of *tied* constraints is the union of the outputs of every possible ranking of those constraints (Pesetsky 1997: 12).

When a set of tied constraints occurs at the bottom of a hierarchy, the effect is the same as that of a set of indifferently ranked constraints. An example of this is shown in (21), where the tie between LE(CP) and TEL results in three-way free variation.

(21) Simple relative clauses in English (adapted from Pesetsky 1997: 13)

Candidates	REC	LE(CP)	TEL
A. *the person [CP who that Mary invited t to the party]		*	*!
B. the person [CP who ~~that~~ Mary invited t to the party] **[on ranking: TEL>>LE(CP)]** ☞		*	
C. the person [CP ~~who~~ that Mary invited t to the party] **[on ranking: LE(CP)>>TEL]** ☞			*
D. the person [CP ~~who~~ ~~that~~ Mary invited t to the party] **[on ranking: TEL>>LE(CP)]** ☞		*	

This tableau calculates which elements of the structure 'the person [CP who that Mary invited t to the party]' are to be pronounced. The constraints involved are REC[overability], which requires the pronunciation of syntactic units that have semantic content and lack local antecedents; L[eft]E[dge]CP, which (in the formulation in effect at this point in Pesetsky's paper) requires the leftmost pronounced element of CP to be its head (in this case, *that*); and TEL[egraph], which penalizes the pronunciation of function words. Unpronounced words in each candidate are ~~struck through~~.

In this case, ranking LE(CP) over TEL favours candidate C, while the opposite ranking produces a tie between B and D. Neither ranking allows A. This situation is compatible with the notion that LE(CP) and TEL are indifferently ranked, and thus taken in random order by the algorithm. When the algorithm takes LE(CP) first, it produces C; when it takes TEL first, it produces a structure in which *that* is unpronounced, or [+silent] in the notation used by Broihier (1995), and *who* is effectively underspecified for the feature [±silent]. Both B and D would be compatible with such a structure.

Pesetsky's tied constraints differ from indifferently ranked constraints, however, when other constraints are ranked below them, as in (22):

(22) adapted from Pesetsky (1997: 20)

Candidates	REC	LE(CP)	TEL	DCP
A. [That the world is round] is believed by everyone] ☞			*	
B. *[~~That~~ the world is round] is believed by everyone]		*		*!

The new constraint, D[eletion in]CP, penalizes the deletion of the head of a CP that is not a complement. Under Pesetsky's definition of tied constraints, both candidates survive LE(CP) and TEL because each candidate is preferred by one ranking of the two constraints, and so the choice between them is made by DCP. (In this particular example, equally ranked constraints would have the same effect. The constraint hierarchy in (22), however, would still choose candidate A even if it contained two violations of TEL, because A would still be allowed by one of the possible rankings of the tied constraints.) If LE(CP) and TEL were indifferently ranked, then the resulting grammar would allow both A and B. The algorithm would operate as follows. On occasions when the algorithm chose to process LE(CP) first, *that* would be required to be pronounced, and the algorithm would generate form A. However, when the algorithm selected TEL first, the output would be required to leave function words unpronounced, and both LE(CP) and DCP would be violated for the sake of this requirement, resulting in form B. Tied constraints (in Pesetsky's theory) and indifferently ranked constraints (as treated by the algorithm in (1)) are similar in that a group of tied or indifferently ranked constraints produces all the possible outputs of any ranking of those constraints. Where they differ is in *when* they produce these outputs: tied constraints yield all the outputs together, so that they may compete with one another on lower-ranked constraints, while indifferently ranked constraints produce their various outputs in separate calculations. Because the algorithm in (1) is not based on competition between candidates, it does not lend itself to tied constraints of the sort proposed by Pesetsky.

The reasons for limiting the range of allowable ranking relations among constraints are less compelling than the reasons for disallowing sympathy, but they are significant nonetheless. First, reducing the number of possible constraint rankings limits the task of the learner in acquiring an OT grammar. While a comprehensive theory of acquisition is well beyond the scope of this paper, the algorithm in (1) is compatible with some form of acquisition by constraint demotion. Tesar and Smolensky's (1998) constraint demotion algorithm depends on the comparative harmonic evaluation that is central to conventional Optimality Theory: the learner looks for a constraint in the hypothesized grammar that assigns a fatal violation to an observed form, and demotes that constraint accordingly. Clearly, this method cannot be duplicated exactly in a theory that does not involve explicit comparison between candidates. However, the algorithm in (1) does permit a closely analogous method. Given an observed form not produced by the learner's hypothesized grammar, the learner can find the point in the calculation where the specifications of the structure being calculated by the hypothesized grammar become incompatible with the observed form. For example, suppose a learner acquiring Czech has a grammar that produces [*szvonem] instead of the observed [sezvonem]. The learner can go through the calculation of [*szvonem] and find the place where the possibility of an additional segment between the [s] and the [z] is eliminated. This will turn out to be the point in the calculation where DEP is incorporated, and so DEP is the constraint that needs to be demoted.

The other reason for limiting constraint rankings is the usefulness of trying the most restrictive version of the theory first. As mentioned in section 1, Prince and Smolensky (1993) disapprove of the notion of restriction imposed for its own sake. However, if we start with a highly constrained theory, we can add power to it only as needed, and by proceeding in this manner we will discover precisely how much power is necessary, and why. For example, suppose we begin with the assumption that constraints cannot be equally ranked. When we find a situation in which two constraints appear to have exactly the same force in a grammar (as in Truckenbrodt's analysis of Bengali), we will then be led to ask whether these two constraints might be collapsed into one—in which case we would expect them to act as a single constraint cross-linguistically. If they do not, then we may be forced to admit the possibility of equally ranked constraints, but we will know exactly why we are doing so. However, if we allow equally ranked constraints from the beginning, their presence in the theory will remain nothing more than a stipulation.

In the end, equally ranked constraints, tied constraints, or other variations may turn out to be necessary, and I would not rule them out absolutely. Unlike the sympathy analysis of Tiberian Hebrew, the analyses advanced by Truckenbrodt and by Pesetsky offer genuinely new ways of looking at the data, provoked by the unique character of Optimality Theory. In the case of Pesetsky's tied constraints, the restrictions imposed by the algorithm in (1) may simply be irrelevant; since Pesetsky is applying Optimality Theory to the limited task of determining which elements in a structure to pronounce, his candidate sets are finite, and so comparative harmonic evaluation is a computationally viable method for his purposes. The purpose of the present paper, however, is to propose an inital formulation of a restrictive general algorithm for making optimality calculations, and this end is best served by disallowing tied and equally ranked constraints.

4. Implementing the algorithm

Since the purpose of this paper is to propose a computable version of Optimality Theory, it is incumbent upon the proposed grammar to compute. The program in the appendix represents a first attempt at implementing the algorithm in (1), using Turing, a computer language developed at the University of Toronto. Turing is similar to the better-known Pascal, but it includes a wider range of built-in functions and data structures. A complete account of Turing syntax may be found in Holt (1993).

The program in the appendix implements the constraints from the Czech epenthesis example in Figure 1: MAX, IDENT, $*C_iC_i$, DEP, and X≈e. The output form being calculated is represented as a linked list, with each item in the list corresponding to a segment or possible string of segments. The properties associated with each item include quality, lower limit, upper limit, and input correspondent. For example, a possible list item might stand for 'at least zero and at most five instances of a segment of unspecified quality, coindexed with the third segment of the input.' In the current implementation, quality is simply represented by alphabetic characters, and the program uses a lookup table to determine whether a sequence violates $*C_iC_i$. A more sophisticated implementation would allow for the separate representation of individual feature values such as Coronal, Dorsal, [±continuant], etc. In addition, each list item contains boolean variables that serve as diacritic features indicating restrictions on the item—for instance, the boolean variable **faith**, whose value is set to **true** by IDENT, indicates that if a list item acquires an input correspondent, it must also acquire the features of that correspondent. These features are a (somewhat inelegant) mechanism for ensuring that the restrictions imposed by higher-ranked constraints are respected by the actions of lower-ranked constraints. In an ideal implementation, such restrictions would be encoded only in the constraints that impose them; however, the present implementation cannot provide such modularity.

The constraints are implemented as procedures that operate on the output, increasing the degree to which it is specified. To ensure that a lower-ranked constraint does not introduce violations of a higher-ranked one, after each new constraint is incorporated, the procedures associated with the higher-ranked constraints are run again. If the constraints are ranked 1>>2>>3>>4>>5, then the corresponding procedures run in the order 1; 2, 1; 3, 1, 2; 4, 1, 2, 3; 5, 1, 2, 3, 4. Some of the procedures are much simpler than others. For example, the procedure that corresponds to DEP simply minimizes the number of output segments that lack input correspondents, while the procedure for MAX must carry out a complex set of calculations to determine whether it can enforce the presence of input material in the output. Here again, a greater degree of modularity would be desirable; in the current implementation, MAX must see whether the segments it is trying to introduce are compelled to be faithful to their input correspondents, and, if so, whether the features of their correspondents would cause a violation of $*C_iC_i$, and if so, whether the possibility of separating them by an epenthetic segment has been ruled out.

Constraints should not have this sort of knowledge about other constraints. However, every markedness constraint implies a fixed number of repair strategies for its satisfaction. $*C_iC_i$ can be satisfied by deletion, by epenthesis, by dissimilation, by coalescence, or by metathesis. Each of these strategies is the negation of a faithfulness constraint: deletion and coalescence violate MAX, epenthesis violates DEP, dissimilation violates IDENT, and metathesis (not considered here) violates LINEARITY. An ideal version of the program in the index would keep track of which strategies were available, and allow each markedness constraint to impose the optimal strategy without having constraints refer to one another directly. In such an implementation of the algorithm in (1), Optimality Theory might come to look more like the theory of constraints and repair strategies proposed by Paradis (1988).

The program stops once all constraints have been incorporated. Since lower-ranking constraints often have no bearing on the calculation of an output form, it might be more efficient to allow the algorithm to stop as soon as it has produced a fully specified form; since constraints cannot undo specifications, there would be no reason to continue the calculation once such a form has been reached. However, the line between phonology and phonetics, especially in OT, is not clearly drawn. At this point, it is difficult to say exactly what constitutes a 'fully specified' form.

When run, the program in the appendix produces the correct output for /szvonem/ under each of the 120 possible rankings of the five constraints. (For half of these rankings, the correct output is [eeeeeee], because X≈e dominates IDENT; the other 60 represent more plausible grammars for natural languages.) While the operation of the program is not ideal, its success suggests that the algorithm is indeed computable, and its shortcomings indicate interesting directions for refinements.

5. Conclusion: In praise of inadequacy

Chomsky (1973) defines three levels of adequacy that grammars may aspire to: *observational adequacy*, or the ability to generate all and only the grammatical sentences of a language; *descriptive adequacy*, or the ability to assign the correct structural descriptions to all sentences of a language; and *explanatory adequacy*, or the ability to explain how competence in a language can be acquired. While none of these levels of adequacy has yet been fully attained for any known language, most comparisons of linguistic theories are phrased in terms of the difference between descriptive and explanatory adequacy. A typical argument put forward in favour of various theories is that they are 'more explanatory' than their rivals. Explanatory adequacy is undoubtedly a worthy goal; in fact, it is the ultimate aim of generative linguistics, and in section 1, I argued for computability on the grounds that it is a necessary component of an explanatorily adequate theory of language. In this section, however, I wish to put forward the notion that the principal virtue of the algorithm proposed in this paper is that it is not even observationally adequate.

Section 3 of this paper explains how the algorithm in (1) disallows various devices that have been proposed as additions to Optimality Theory.

For instance, the algorithm rules out the possibility of sympathy constraints, thereby rendering the theory highly resistant to opacities resulting from non–surface-apparent generalizations. Such opacities are widely attested among the languages of the world. However, the original analyses of these languages are, for the most part, framed in terms of theories for which opacity presents no obstacle. More recent work, such as Bakovic (1998) and Balcaen and Hall (1999), indicates that some supposed opacities are susceptible to reanalysis in non-opaque terms. Without the impetus provided by Optimality Theory—or some other theory resistant to opacity—such counter-analyses might never have been proposed.

Because non–surface-true generalizations are so widely reported, I believe it is extremely likely that opacity is genuinely a property of human language—perhaps even, as Idsardi (1999) claims, its fundamental property. If this is the case, then (given a few limiting assumptions about the power and formulation of constraints), the algorithm in (1) cannot possibly serve as the basis for an observationally adequate theory of language. However, it is only by attempting to construct such a theory that we can discover to what extent language is necessarily opaque, rather than merely opaque by hypothesis. Similarly, by attempting to construct a theory without equally ranked constraints, we can discover whether such constraints are genuinely necessary, and, if so, where. The algorithm in proposed in this paper is almost certain to fail as a model of how language works, but it will fail in informative ways.

References

Bakovic, E. 1998. Deletion, insertion, and symmetrical identity. Ms., Harvard and Rutgers.

Balcaen, M. Jean, and Daniel Currie Hall. 1999. Tiberian Hebrew: Opaque, or simply abstract? Paper presented at the Montréal-Ottawa-Toronto Phonology Workshop, McGill University, February 1999.

Broihier, Kevin. 1995. Optimality Theoretic rankings with tied constraints: Slavic relatives, resumptive pronouns and learnability. Ms., MIT. ROA #46.

Chomsky, Noam. 1973. *The Logical Structure of Linguistic Theory.* New York: Plenum.

de Lacy, Paul. 1999. Sympathetic stress. Ms., University of Massachustetts, Amherst. ROA #294.

Eisner, Jason. Doing OT in a straitjacket. Paper presented at the University of Toronto.

Ellison, T. Mark. 1995. Phonological derivation in Optimality Theory. ROA#75.

Frank, Robert, and Giorgio Satta. 1998. Optimality theory and the generative complexity of constraint violability. ROA #228.

Gesenius, W. 1910. *Gesenius' Hebrew Grammar*. Edited by E. Kautzsch and A.E. Cowley. 28th ed. Oxford University Press.

Hall, Daniel Currie. 1998. Contrastive specification for West Slavic voicing assimilation. Paper presented at the Montréal-Ottawa-Toronto Phonology Workshop, University of Ottawa, February 1998.

Hammond, Michael. 1997. Parsing Syllables: Modelling OT computationally. ROA #222.

Heiberg, Andrea. 1999. Features in Optimality Theory: A computational model. Doctoral dissertation, University of Arizona. ROA #321.

Holt, R.C. 1993. *Turing Reference Manual*. Toronto: Holt Software Associates.

Idsardi, William J. 1997. Sympathy creates chaos. Ms., University of Delaware, Newark.

———. 1999. Phonological opacities. Paper presented at the University of California at Irvine, May 7, 1999.

Jun, Jongho. 1998. Generalized Sympathy. Paper presented at NELS 29 at the University of Delaware. ROA# 297.

Karttunen, Lauri. 1998. The proper treatment of optimality in computational phonology. ROA #258

Karvonen, Dan, and Adam Sherman. 1997. Opacity in Icelandic: A sympathy account. Paper presented at NELS 28 at the University of Toronto.

Malone, J.L. 1993. *Tiberian Hebrew Phonology*. Winona Lake, Ind.: Eisenbrauns.

McCarthy, John J. 1998. Sympathy and phonological opacity. Ms., University of Massachusetts, Amherst. ROA #252.

McCarthy, John, and Alan Prince. 1995. Faithfulness and reduplicative identity. *University of Massachusetts Occasional Papers* 18: 249–384. ROA #60.

Paradis, Carole. 1988. On constraints and repair strategies. *The Linguistic Review* 6: 71–97.

Pesetsky, David. 1997. Some optimality principles of sentence pronunciation. Proceedings of the "Good Enough" conference at MIT. ROA #184.

Prince, A. 1975. The phonology and morphology of Tiberian Hebrew. Doctoral dissertation, MIT.

Prince, Alan, and Paul Smolensky. 1993. Optimality Theory: Constraint interaction in generative grammar. Technical Report #2 of the Rutgers Centre for Cognitive Science. Piscataway, NJ: Rutgers University.

Pullum, Geoffrey. 1976. The Duke of York Gambit. *Journal of Linguistics* 12, 83–102.

Tesar, Bruce, and Paul Smolensky. 1998. Learnability in Optimality Theory. *Linguistic Inquiry* 29: 229–268.

Truckenbrodt, Hubert. 1997. Correspondence of syntactic and phonological phrases. Ms., Rutgers University.

Appendix: The Program

```
var output: collection of forward segment
type char: string(1)
type segment:
  record
    quality: char
    minim: int
    maxim: int
    indx: int
    nexts: pointer to output
    prevs: pointer to output
    ocp: boolean
    faith: boolean
    lic: boolean
  end record
const input:="szvonem"
const infinity:=2**30

function disp(value: int): string
  var temp: string
  if value<0 then
    temp:="-"
  elsif value<infinity then
    temp:=intstr(value)
  else temp:="*"
  end if
  result temp
end disp

function counterpart(seg: char): char
  var temp: char
  case seg of
    label "p": temp:="b"
    label "b": temp:="p"
    label "t": temp:="d"
    label "d": temp:="t"
    label "k": temp:="g"
    label "g": temp:="k"
    label "f": temp:="v"
    label "v": temp:="f"
    label "s": temp:="z"
    label "z": temp:="s"
    label "x": temp:="h"
    label "h": temp:="x"
    label:     temp:="%"
  end case
  result temp
end counterpart

function cons(value: char): boolean
  var temp:=true
  if index("pbtdkgfvszxh", value)=0 then
    temp:=false
  end if
  result temp
end cons

function mcompatible(value: char, seg: pointer to
output): boolean
  var temp:=true
  var p, n: char
  const c:=counterpart(value)
  if output(seg).ocp=true and cons(value) then
    if output(seg).prevs not= nil(output) then
      p:=output(output(seg).prevs).quality
      if output(output(seg).prevs).faith and
       output(output(seg).prevs).indx>0 and p=" "
         and (output(seg).maxim<2 or output(seg).lic)
then
         p:=input(output(output(seg).prevs).indx)
      end if
    else
      p:=" "
    end if
    if output(seg).nexts not= nil(output) then
      n:=output(output(seg).nexts).quality
      if output(output(seg).nexts).faith and
       output(output(seg).nexts).indx>0 and n=" "
         and (output(seg).maxim<2 or output(seg).lic)
then
         n:=input(output(output(seg).nexts).indx)
      end if
    else
      n:=" "
    end if
    if p=value or n=value or p=c or n=c then
      temp:=false
    end if
  end if
  result temp
end mcompatible

function compatible(value: char, seg: pointer to
output): boolean
  var temp:=mcompatible(value, seg)
  var p, n: char
  const c:=counterpart(value)
  if output(seg).faith and output(seg).indx>-1 then
    if output(seg).indx=0 then
      temp:=false
    elsif value not= input(output(seg).indx) then
      temp:=false
    end if
  end if
  result temp
end compatible
```

```
function half(quantity: int): array 1..2 of int
  var temp: array 1..2 of int
  if quantity=infinity then
    temp(1):=infinity
    temp(2):=infinity
  elsif quantity>0 then
    temp(1):=floor((quantity-1)/2)
    temp(2):=ceil((quantity-1)/2)
  else
    temp(1):=0
    temp(2):=0
  end if
  result temp
end half

procedure maxseg(var root: pointer to output)
  var unparsed: array 1..length(input) of int
  for lcv: 1..length(input)
    unparsed(lcv):=lcv
  end for
  var current:=root
  loop
    exit when current=nil(output)
    if output(current).indx>0 and
    output(current).indx<=length(input) then
      unparsed(output(current).indx):=0
    end if
    current:=output(current).nexts
    for lcv: 1..length(input)-1
    if unparsed(lcv)=0 then
      for ilcv: lcv..length(input)-1
        unparsed(ilcv):=unparsed(ilcv+1)
      end for
    end if
  end for
  var lcv:=1
  current:=root
  loop
    exit when lcv>length(input)
    exit when unparsed(lcv)=0
    exit when current=nil(output)
    if output(current).indx=0 and
    output(current).maxim>0 and
    (output(current).faith=false or
    mcompatible(input(unparsed(lcv)), current)) then
      var before, after: pointer to output
      new output, before
      new output, after
      output(before).prevs:=output(current).prevs
      output(before).nexts:=current
      output(after).prevs:=current
      output(after).nexts:=output(current).nexts
      if output(current).prevs not= nil(output) then
        output(output(current).prevs).nexts:=before
      else
        root:=before
      end if
      if output(current).nexts not= nil(output) then
        output(output(current).nexts).prevs:=after
      end if
      output(before).quality:=output(current).quality
      output(after).quality:=output(current).quality
      output(before).indx:=0
      output(after).indx:=0
      output(before).faith:=output(current).faith
      output(after).faith:=output(current).faith
      output(before).lic:=output(current).lic
      output(after).lic:=output(current).lic
      output(before).ocp:=output(current).ocp
      output(after).ocp:=output(current).ocp
      output(before).minim:=
            half(output(current).minim)(1)
      output(after).minim:=
            half(output(current).minim)(2)
      output(before).maxim:=
            half(output(current).maxim)(1)
      output(after).maxim:=
            half(output(current).maxim)(2)
      output(current).indx:=unparsed(lcv)
      output(current).prevs:=before
      output(current).nexts:=after
      output(current).minim:=1
      output(current).maxim:=1
      lcv:=lcv+1
      current:=output(current).nexts
    elsif output(current).faith and
    not(mcompatible(input(unparsed(lcv)),
            current)) then
      lcv:=lcv+1
    else
      current:=output(current).nexts
    end if
  end loop
  current:=root
  loop
    exit when current=nil(output)
    if output(current).indx=0 then
      output(current).indx:=-1
    end if
    current:=output(current).nexts
  end loop
end maxseg

procedure depseg(root: pointer to output)
  output(root).lic:=true
  if output(root).indx<0 then
    output(root).maxim:=output(root).minim
  end if
  if output(root).nexts not= nil(output) then
    depseg(output(root).nexts)
  end if
end depseg
```

```
procedure ve(root: pointer to output)
  if output(root).quality=" "
  and compatible("e", root) then
    output(root).quality:="e"
  elsif output(root).quality=" "
  and output(root).faith and output(root).indx>0 and
    not(compatible(input(output(root).indx),  root))
then
      output(root).quality:="e"
  end if
  if output(root).nexts not= nil(output) then
    ve(output(root).nexts)
  end if
end ve

procedure ident(root: pointer to output)
  if output(root).indx>-1 then
    output(root).faith:=true
  end if
  if output(root).indx>0
  and output(root).quality=" " then
    if compatible(input(output(root).indx), root) then
      output(root).quality:=input(output(root).indx)
    end if
  end if
  if output(root).nexts not= nil(output) then
    ident(output(root).nexts)
  end if
end ident

procedure cici(root: pointer to output)
  var current:=root
  var mor, ubm, ges, vrg: pointer to output
  var cq, uq, vq: char
  loop
    exit when current=nil(output)
    output(current).ocp:=true
    mor:=output(current).nexts
    ges:=output(current).prevs
    cq:=output(current).quality
    if cq=" " and output(current).faith and
    output(current).indx>0 then
      cq:=input(output(current).indx)
    end if
    if cons(cq) then
      if mor not= nil(output) then
        ubm:=output(mor).nexts
        if ubm not= nil(output) then
          uq:=output(ubm).quality
          if uq=" " and output(ubm).faith and
           output(ubm).indx>0 then
             uq:=input(output(ubm).indx)
           end if
          if (uq=cq or uq=counterpart(cq)) and
             (output(mor).minim=0 and
             output(mor).maxim>=1) then
             output(mor).minim:=1
           end if
        end if
      end if
      if ges not= nil(output) then
        vrg:=output(ges).prevs
        if vrg not= nil(output) then
          vq:=output(vrg).quality
          if vq=" " and output(vrg).faith and
           output(vrg).indx>0 then
             vq:=input(output(vrg).indx)
           end if
          if (vq=cq or vq=counterpart(cq)) and
           (output(ges).minim=0 and
           output(ges).maxim>=1) then
             output(ges).minim:=1
           end if
        end if
      end if
    end if
    current:=mor
  end loop
end cici

procedure prune(var root: pointer to output)
  var current:=root
  loop
    exit when current=nil(output)
    if output(current).maxim=0 then
      if output(current).prevs=nil(output) then
        root:=output(current).nexts
      else
        output(output(current).prevs).nexts:=
                        output(current).nexts
      end if
      if output(current).nexts not= nil(output) then
        output(output(current).nexts).prevs:=
                        output(current).prevs
      end if
    end if
    current:=output(current).nexts
  end loop
end prune

procedure display(root: pointer to output)
  var current:=root
  put "Quality  Index  Minimum  Maximum"
  loop
    exit when current=nil(output)
    put "    "+output(current).quality+"    "..
    put "  "..
    put "  "+disp(output(current).indx)+"  "..
    put "  "..
    put "    "+disp(output(current).minim)+"    "..
    put "  "..
    put "    "+disp(output(current).maxim)+"    "
    current:=output(current).nexts
  end loop
end display
```

```
procedure apply(constraint: char,
                                    var root: pointer to output)
  case constraint of
    label "m": maxseg(root)
    label "d": depseg(root)
    label "i": ident(root)
    label "c": cici(root)
    label "e": ve(root)
  end case
  prune(root)
end apply

procedure generate(ranking: string,
                                               var root: pointer to
output)
  const sum:=length(ranking)
  for lcv: 1..sum
    apply(ranking(lcv), root)
    if lcv>1 then
      for ilcv: 1..lcv-1
        apply(ranking(ilcv), root)
      end for
    end if
  end for
end generate

const ranking:="micde"

        var root: pointer to output
        new output, root
        output(root).nexts:=nil(output)
        output(root).prevs:=nil(output)
        output(root).indx:=0
        output(root).maxim:=infinity
        output(root).minim:=0
        output(root).quality:=" "
        output(root).ocp:=false
        output(root).faith:=false
        output(root).lic:=false
        generate(ranking, root)
        put "Constraint Ranking: "+ranking
        display(root)
        put ""
```